Terraform Training

LEVEL 1: Terraform Beginner

Agenda

01	What Infrastructure as Code (IaC) is and why it matters
02	Common problems with manual and script-based infrastructure management
03	The difference between Imperative and Declarative approaches
04	Why Terraform is the industry standard for IaC
05	Basic Terraform architecture and workflow

What is Infrastructure as Code (IaC)?

> Infrastructure as code is a way to ***automatically*** **manage infrastructure** with **configuration** files rather than ***manually*** through a *web user interface* or ***semi-automatically*** through *command line* scripts.

Manage = **C**reate/**R**ead/**U**pdate/**D**elete = **CRUD**

Infrastructure = Compute/Storage/Networking resources

- Compute (AWS EC2 instances, Containers)
- Storage (S3, Azure Storage, Disks)
- Networking (VPC, Subnets, Load Balancers)
- Databases (RDS, DynamoDB, SQL)

Ways to manage infrastructure

Let's explore common approaches to infrastructure management and their limitations:

1. **Manual Web page (AWS Console) approach**
    - Good for quick one-off tasks
    - Poor for large-scale environments

2. **Command Line scripts approach**
    - Faster than console for repetitive tasks
    - Hard to maintain and scale

Scenario 1: Using *AWS Console* to manage infrastructure.

Example: Launch an EC2 instance, then change instance type and re-launch.

Why we do it?

- Quick for simple one-off resources
- No coding required

What can go wrong?

- Time-consuming for multiple resources
- Difficult to track changes
- Prone to human errors
- Not reproducible across environments
- No version control

Scenario 2: Using *Command Line* to manage infrastructure.

Example: Write scripts to launch EC2 instances and modify their types.

Why we do it?

- Faster than AWS Console once scripts are developed
- Automates repetitive tasks

What can go wrong?

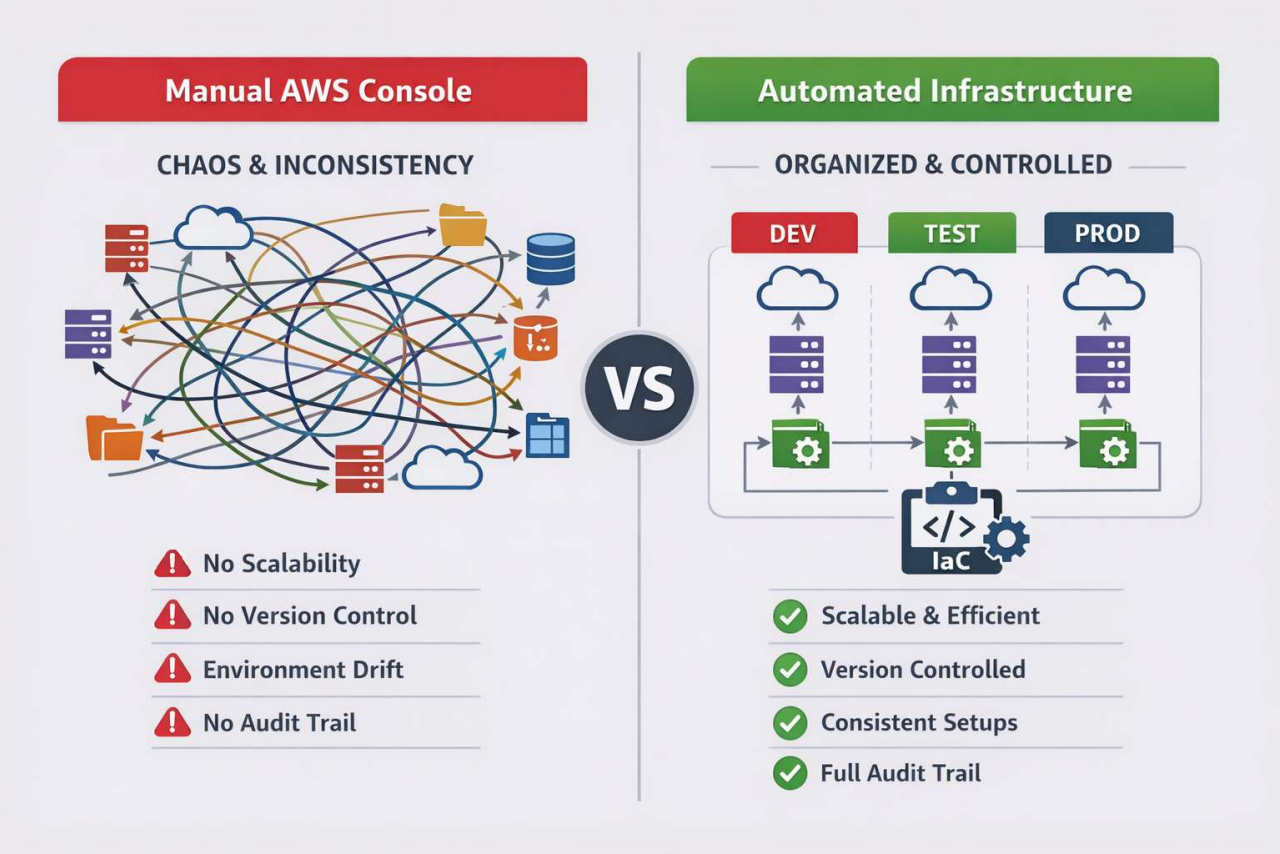
- No automatic dependency handling
- Difficult error handling
- Not idempotent (running twice creates duplicates)
- Hard to validate compliance

What's Missing?

Both approaches have significant limitations:

AWS Console Problems:

- Not scalable beyond small setups
- No version control for infrastructure
- Hard to replicate across Dev, Test, Prod
- No audit trail of changes



- No automatic dependency resolution
- Complex error handling
- Not truly idempotent
- Scripts themselves are not properly audited

What we need:

- Automatic dependency handling
- **Declarative approach** (define *what*, not *how*)
- State tracking of infrastructure
- Idempotent execution
- Predictable changes

The Solution: Declarative IaC

Instead of writing step-by-step instructions (imperative), we:

1. **Describe the desired end state** (declarative)
2. Let the IaC tool figure out how to get there
3. The tool handles dependencies, errors, and state automatically

Analogy: Restaurant Order

You say: "I want a Margherita pizza"
Chef decides how to prepare it
You don't give cooking instructions

Terraform = Chef

Imperative vs. Declarative IaC

Imperative = Step-by-step commands

Declarative = Desired final state

Approach	**Example**	**Benefit**	**Limitation**
Imperative	Scripts	Control	Complex
Declarative	Terraform	Simple	Less control

Imperative - step by step sequence of commands

Example: Changing instance type

1. Launch an EC2 with m5.large type
2. Wait until instance is up and running
3. Find the instance you previously launched
4. Stop instance
5. Change instance type to m5.2xlarge
6. Start instance

Problems:

- Order matters
- Breaks if someone manually changes infrastructure

Declarative - define how the infrastructure should look

Example: Changing instance type

1. Create config file with `instance_type = "m5.large"`
   →Run terraform apply
2. Modify config to `instance_type = "m5.2xlarge"`
   →Run terraform apply

Advantages:

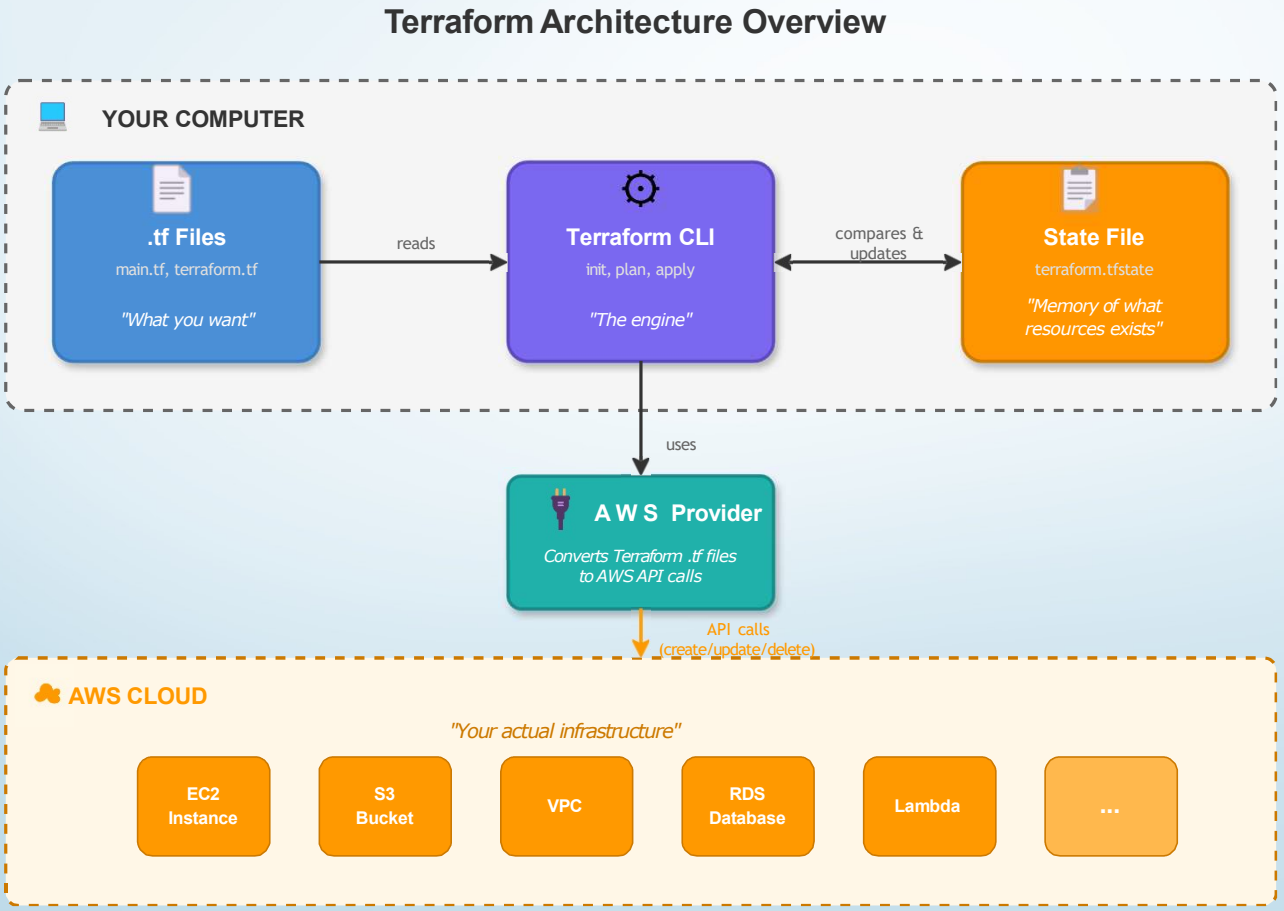
Idempotent (safe to run multiple times)

Handles dependencies automatically

Self-documenting

Gitversion controlled since it's a text

Basic Terraform architecture overview



Key Terraform Components

- **Configuration Files** (.tf files) - What you write (desired state)
- **Terraform Command Line** (terraform command) - Engine that processes your configuration
- **Providers** - Plugins that talk to cloud platforms (AWS, Azure, etc.)
- **State File** - Terraform's memory of what currently exists
- **Resources** - The actual infrastructure created in your cloud

The Terraform Workflow

Terraform follows a simple, repeatable workflow:

1. **Define** - Create `.tf` configuration files
2. **Init** - `terraform init` - Initialize working directory and download providers
3. **Plan** - `terraform plan` - Show changes required by current configuration
4. **Apply** - `terraform apply` - Execute actions to create, update, or destroy infrastructure
5. **Verify** - Use `terraform show` or `terraform state list` to inspect state
6. **Modify** - Change configuration files as needed (repeat from step 3)
7. **Destroy** - `terraform destroy` - Remove all managed infrastructure

Key Takeaways

IaC automates infrastructure management through configuration files

Declarative approach (Terraform) beats imperative (scripts) for scalability

Terraform is the industry-standard multi-cloud IaC tool

Core workflow: Define →Init →Plan →Apply →Destroy

Terraform handles **dependencies**, **state**, and **idempotency** automatically

Knowledge Check 1

What does "idempotent" mean in the context of Terraform?

A) Running the same operation once or multiple times produces the same result

B) Each run creates new duplicate resources

C) The operation can only be run once

D) The operation fails if run more than once

Knowledge Check 2

What is the correct order of the basic Terraform workflow?

A) Apply ⟶Init ⟶Plan ⟶Destroy

B) Plan ⟶Init ⟶Apply ⟶Destroy

C) Init ⟶Plan ⟶Apply ⟶Destroy

D) Destroy ⟶Init ⟶Plan ⟶Apply

Knowledge Check 3

- Your team needs to track all infrastructure changes for regulatory compliance. Which Terraform advantage helps with this?

A) Terraform is free

B) Configuration files can be version controlled in Git

C) Terraform runs faster than scripts

D) Terraform doesn't require permissions